.NET programok minőségi mutatóinak javítása axióma alapú automatizált teszteléssel

Doktori értekezés tézisei

Biczó Mihály

Témavezető:
Dr. Porkoláb Zoltán

Eötvös Loránd Tudományegyetem
Informatika Doktori Iskola
Iskolavezető: Dr. Benczúr András

Az informatika alapjai és módszertana doktori program
Programvezető: Dr. Demetrovics János

Budapest, 2012

A projekt az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósul meg (a támogatás száma TÁMOP 4.2.1./B-09/1/KMR-2010-0003).

1. Bevezetés

A dolgozatban bemutatott eredmények .NET környezetben készült programok minőségi mutatóinak javítására szolgálnak. Bertrand Meyer a programok minőségi mutatóit két csoportra osztotta: a rendszert felhasználói szempontból jellemző külső, illetve a fejlesztői szempontból jellemző belső minőségi mutatókra. Az Eiffel programozási nyelv megalkotásakor abból a feltevésből indult ki, hogy a belső minőségi mutatók jobbá válásából következik a külső minőség javulása is. Az Eiffel nyelvi eszközeinek a segítségével a későn feltárt hibák száma hozzávetőlegesen 30%-kal csökkenthető, amely azt jelenti, hogy a felhasználók ennyivel kevesebb hibával szembesülnek az alkalmazásban.

Értekezésemet – amelyhez a fő motivációt Bertrand Meyer gondolatai és az Eiffel szerződés alapú programozást támogató nyelvi eszközei adták - ezen alapelvek köré építettem fel. Célom egy olyan eszköztár kidolgozása, amelyet a fejlesztők közvetlenül felhasználhatnak a kevesebb hibát tartalmazó (jobb belső minőségű) alkalmazások készítése során. Vizsgálataimhoz a .NET környezetet választottam, részben népszerűsége miatt, részben pedig azért, mert 2002-es megjelenése óta szolgáltatásainak halmaza folyamatosan bővül. Ez alatt a közel tíz év alatt mind a platform, mind pedig annak egyik legnépszerűbb programozási nyelve olyan elemekkel egészült ki, amelyek az egyszerűbb, áttekinthetőbb szerkezetet, a könnyebb olvashatóságot, a kompaktabb leírásokat, vagy éppen a korábbinál kedvezőbb hatékonyságot szolgálják. Olyan újítások ezek, amelyek a fejlesztők számára megkönnyítik a minőségi szoftvertermékek létrehozását. A 2002 óta eltelt idő sem volt azonban elég ahhoz, hogy a C# nyelv az Eiffeléhez hasonló, opcionálisan használható helyességi specifikációs eszköztárat kapjon. Kísérletek zajlottak ugyan a nyelv ilyen irányú kiterjesztésére (például Spec# kísérleti nyelv), a helyességi specifikációs részнyelv azonban sosem került be a C# szabványba.

Az értekezésben bevezetett infrastruktúra az Eiffel nyelv helyességi specifikációs mechanizmusához hasonló: a programról állításokat fogalmazhatunk meg (osztályokhoz és metódusokhoz kapcsolva), amelyeket a futtató környezet a végrehajtás során ellenőriz. Ezen állítások elméleti bizonyítása a gyakorlatban nehézségekbe ütközik, ezért általánosan elfogadott megoldás, hogy azt ellenőrizzük, hogy egy adott végrehajtási úton szereplő értékek kielégítik-e az állításokat. Amennyiben igen, feltételezzük, hogy a program helyesen működik.

Másfelől, a gyakorlatban a specifikáció formalizálása – az implementációs részletek figyelembe vételével - nem triviális feladat. Ezért gyakori megközelítés, hogy programunk helyességét előre meghatározott tesztadatok felhasználásával ellenőrizzük. Ennek a feladatnak az elvégzésére a szoftvermérnöki tudomány számos, ma már széles körben elterjedt eljárást dolgozott ki. Ilyen például az egységtesztelés (unit testing), a teszteset-vezérelt programozás (test-driven development), a szimulált függőségek (mocks, stub objects) használata. A tesztesetek általában az implementáció előtt készülnek el, így írva körül az elvárt működést. A megoldás a gyakorlatban jól használható, strukturáltabb, egyszerűbben módosítható, olvashatóbb komponensek megalkotására ösztönzi a fejlesztőket.

Dolgozatomban a két fent említett módszert - a helyességi specifikációt és az egységtesztelést – egyesítem, nem megtagadva, sokkal inkább kiegészítve az ismertetett alapelveket. A bevezetett helyességi specifikációs infrastruktúrát alkalmassá teszem arra, hogy magasabb szintű összefüggéseket, *axiómákat* kapcsolhassunk a programban szereplő interfészekhez. Az axiómák használatának a célja kettős. Egyfelől, segítségükkel az egységteszteknél kompaktabb, egységesebb, és magasabb absztrakciós szintű specifikációs leírást mellékelhetünk a fogalmakhoz. Amíg az egységtesztek valamely kis programegység - tipikusan metódus – izolált tesztelésére hivatottak, az axiómák metódusok egymással való kölcsönhatásáról fogalmaznak meg állításokat, így alkalmasak az egységtesztek és integrációs tesztek között fennálló koncepcionális szakadék áthidalására.

Másfelől az axiómák tesztesetek generátoraiként is felhasználhatóak. Kidolgoztam egy eljárást, amely az axiómák alapján automatikusan állít elő tesztmetódusokat. Az egységtesztelés és az axiomatikus tesztelés közötti másik lényeges különbség a teszteléshez használt adatok relevanciája. Míg az egységteszteket jellemzően az implementáció előtt, manuálisan, fiktív tesztadatokkal építjük fel; addig az axiómáknak csak a megfogalmazása történik az implementációval egy időben, a tesztmetódus generálása és a tesztadatokkal történő ellátása csak az implementáció után következik be (automatizáltan). Ez a késői összekapcsolás magában hordozza annak a lehetőségét, hogy a fiktív tesztadatok helyett releváns, adekvát tesztadatokat használjunk fel.

Az utolsó kérdéskör, amellyel értekezésemben foglalkozom, a valódi tesztadatok előállításának a problémája. A korábban ismertetett infrastruktúrát összekapcsolom egy

változó szintű naplózó eljárással, így biztosítva azt, hogy a program futása során általában előforduló értékek szolgáljanak az axiómák bemeneteként.

2. A dolgozat felépítése

Az értekezés egy bevezető részből és hat számozott fejezetből áll. Az első fejezetben az alapfogalmakat és a későbbiekben felhasznált infrastruktúra elemeket ismertetem. A második, harmadik, és negyedik fejezetekben tárgyalom dolgozatom három tézisét. Az ötödik fejezetben a tézisek gyakorlati alkalmazására mutatok kidolgozott esettanulmányt, míg a hatodik fejezet a lehetséges további kutatási irányokat jelöli ki.

A dolgozat eszközrendszerének áttekintése

Értekezésem első fejezetében ismertetetem azokat az eljárásokat és eszközöket, amelyekre a későbbiekben rendre hivatkozom. Röviden áttekintem a közismert szoftverminőségi mutatókat, elvégzem csoportosításukat a felhasználói és fejlesztői nézetrendszer szerint (1.1. Szoftverminőségi mutatók).

Ismertetem az Eiffel helyességi specifikációs nyelvének alapelemeit, megállapítom, hogy ez a helyességi specifikációs nyelv alkalmas programok belső minőségi mutatóinak javítására. Azt az implicit feltételezést, amely szerint a belső minőségi mutatók javulása pozitív hatással van a külső minőségi mutatókra is, a szakirodalom alapján igaznak fogadom el (1.2. Programok helyessége).

Röviden bemutatom az aspektusorientált programozásból felhasznált alapelemeket (1.3. Az aspektusorientált programozásról). Ismertetem a .NET programozási környezet főbb szolgáltatásait (1.4. A .NET keretrendszer áttekintő ismertetése). Az értekezés szempontjából kiemelkedő jelentőséggel bírnak a deklaratív meta-adat leírást lehetővé tevő *attribútumok*, valamint az ezek futási idejű feltárását és feldolgozását biztosító *reflexió* (reflection) mechanizmusa. A .NET-féle megoldást összehasonlítom a Java platform által kínált eszközökkel. A nagyfokú hasonlóság miatt az infrastruktúra minden eleme implementálható, ezért a dolgozatban közölt eredmények átvihetők a Java platformra is.

A fejezet utolsó részében a *függőségek injektálásának* (dependency injection) különféle változataiba vezetem be az olvasót (1.5. Függőségek injektálása). A függőségek injektálása tervezési minta Martin Fowlertől származik, és az elmúlt néhány évben - az üzleti alkalmazások fejlesztésénél mutatott kiváló alkalmazhatóságának köszönhetően - igen népszerűvé vált. Az eszköz értekezésem szempontjából azért jelentős, mert segítségével megvalósítható az aspektusorientált programozás. A helyességi specifikációs

infrastruktúra felépítésénél a függőségek injektálásán alapuló aspektusorientált programozást használom fel.

A programhelyesség-ellenőrzés alapjai .NET környezetben

Dolgozatom első tézisfejezetében (2. fejezet) a helyességi specifikációs infrastruktúrát építem fel. A hatékonysági és rugalmassági szempontból legmegfelelőbb megoldás kiválasztása érdekében összehasonlítom azokat a technikai lehetőségeket, amellyel az aspektusorientált programozást szimulálhatjuk a .NET platformon (2.2. Kapcsolódó kutatások és áttekintés). Az elemzés alapján arra következtettem, hogy a függőségek injektálásán és dinamikus helyettes (dynamic proxy) generáláson alapuló aspektusorientált környezet alkalmas az ipari felhasználásra.

Az értekezésben a Castle MicroKernel függőség injektáló komponens segítségével oldottam meg az aspektusorientált viselkedés adaptációját. A komponens működése a következő: a függőség injektáló konténerbe (dependency injection container) típusokat, illetve a hozzájuk tartozó híváselfogást végző objektumokat (interceptor) regisztrálhatunk. Amikor a regisztrált eredeti típusból példányosítunk (a konténer segítségével), a létrejövő objektum tényleges (dinamikus) típusa egy automatikusan generált helyettes típus lesz. Ebben az eredeti kód meghívása mellett helyet kap a híváselfogást végző objektumban megvalósított eseménykezelők futtatása is.

Az infrastruktúra teljessé tételéhez elkészítettem egy saját helyességi specifikációs nyelvet (2.5. A megvalósítás részletei). Ez a specifikációs nyelv - amely egy szakterület-specifikus leíró nyelv (domain specific language) - C# jellegű szintaxissal teszi lehetővé egyszerű elő- és utófeltételek, illetve invariáns állítások megfogalmazását. A helyességi specifikációs állításokat attribútumok segítségével kapcsolom a metódusokhoz és osztályokhoz.

Az attribútumok feltárása futási időben történik, futási időben jutunk hozzá a helyességi specifikáció szöveges reprezentációjához. Ebből a szöveges reprezentációból kell végrehajtható kódot, esetünkben lambda kifejezéseket előállítani. Ehhez egy lexikai és szintaktikai elemző segítségével felépítünk egy kifejezésfát, amelyből a végrehajtható kód közvetlenül kinyerhető.

1. tézis. Megvalósítottam az Eiffel nyelv szerződés alapú programozási szolgáltatásainak részhalmazát .NET/C# környezetben. A szerződések leírására definiáltam egy helyességi specifikációs nyelvet és a hozzá tartozó elemzőt. A helyességi specifikációból futási időben lambda kifejezéseket generáltam. Az ellenőrzéseket függőségek injektálásán alapuló aspektusorientált technikával oldottam meg.

Releváns publikációk

[14] M. Biczó, K. Pócza, Z. Porkoláb. Runtime access control in C# 3.0 using extension methods. Annales Universitatis Scientiarum Budapestinensis de Rolando Eötvös Nominatae, Sectio Computatorica 30, pp. 41-60, 2009.

[15] M. Biczó, K. Pócza. Generating Functional Implementations of Finite State Automata in C# 3.0. Electronic Notes in Theoretical Computer Science (ENTCS) 238(2), pp. 3-12 2009.

[17] M. Biczó, Z. Porkoláb. Towards Axiom-Based Test Generation In .NET 4 Applications. Annales Universitatis Scientiarum Budapestinensis de Rolando Eötvös Nominatae, Sectio Computatorica 36, pp. 5-22, 2012.

Axiomatikus specifikáción alapuló teszteset generálás

Az értekezés második tézisfejezetében (3. fejezet) bemutatom az axióma alapú teszteset generálást, amely a helyességi specifikációs infrastruktúra egyik lehetséges alkalmazási területe.

Formálisan definiálom a *koncepció* fogalmát. A koncepció egy absztrakt fogalom általános megfogalmazása, programnyelvi megfelelője a sablon interfész. Ennek megfelelően a koncepciónak lehetnek típusparaméterei és absztrakt műveletei. Emellett tartozhat hozzá egy axiómahalmaz is. Az axiómák tipikusan a műveletek egymásra hatására fogalmaznak meg logikai állításokat. Ha a koncepció típusparamétereinek konkrét típusokat feleltetünk meg, és megadjuk az absztrakt műveletek jelentését, a koncepció egy realizációjához jutunk (ez az implementációs folyamat). Az olyan realizációt, amely kielégíti a koncepció axiómáit, modellnek nevezzük (3.3. Elméleti háttér).

A fejezetben megadom, hogy a definíciókban szereplő fogalmak hogyan képezhetők le programnyelvi konstrukciókra (3.4. Magas szintű áttekintés). Az axiómák leírására a legtöbb programozási nyelv nem ad közvetlenül lehetőséget, ezért esetünkben a helyességi specifikációs infrastruktúra alkalmazásával attribútum alapú axiómákat kapcsolok a sablon interfészek által reprezentált koncepciókhoz. Ilyen módon a teljes absztrakt koncepció fogalmat lefedem programnyelvi konstrukciókkal. Megvizsgálom az axiómák és az öröklődés kapcsolatát, az öröklődési hierarchia mentén összegzem az axiómákat és ellenőrzöm azokat. Megoldásom így megfelel a Liskov-féle behelyettesítési elvnek. Az axiómák futtatásához típustól függő adatokat generálok. Ezen adatok ellenében

futtatva az axiómákból előállított tesztmetódusokat tetszőleges realizációról képes vagyok ellenőrizni, hogy lehet-e modell (3.5. A megvalósítás részletei).

2. tézis. A dolgozat második fejezetében kidolgozott infrastruktúra alkalmazásaként eszközt adtam arra, hogy elvont fogalmakhoz a helyesség szükséges feltételeként értelmezhető axiómákat kapcsolhassunk, az axiómák alapján automatizáltan teszteseteket állítsunk elő. Megmutattam azt is, hogy az axiómákból előállított tesztmetódusok milyen módon futtathatók a generált tesztadatokkal. Módszerem nem csak új szoftver termékek létrehozása esetén használható; releváns bemeneti adatok mellett már meglévő rendszerek módosítását is megkönnyíti.

Releváns publikációk

[13] M. Biczó, K. Pócza, I. Forgács, Z. Porkoláb. A New Concept of Effective Regression Test Generation in a C++ Specific Environment. Acta Cybernetica 18 (2008), pp. 481-501 2008.

[17] M. Biczó, Z. Porkoláb. Towards Axiom-Based Test Generation In .NET 4 Applications. Annales Universitatis Scientiarum Budapestinensis de Rolando Eötvös Nominatae, Sectio Computatorica 36, pp. 5-22, 2012.

[91] K. Pócza, M. Biczó, Z. Porkoláb. Towards Effective Runtime Trace Generation Techniques in the.NET Framework. In : Short communication papers proceedings of.NET Technologies 2006, Plzen (Czech Republic), pp. 9-16 2006.

[92] K. Pócza, M. Biczó, Z. Porkoláb. Towards detailed trace generation using the profiler in the.NET Framework. Annales Universitatis Scientiarum Budapestinensis de Rolando Eötvös Nominatae, Sectio Computatorica 30, pp. 21-40 2009.

Releváns tesztadatok előállítása

Értekezésem harmadik tézisfejezetében (4. fejezet) a tesztadatok előállításával foglalkozom. Ehhez definiálom a megbízható tesztadat-halmaz, az adekvát tesztadat-halmaz, illetve a relatív adekvát tesztadat-halmaz fogalmát. Budd és Angluin nyomán megadom az elfogadó eljárás és a generátor eljárás leírását, és megállapítom, hogy egy adekvát tesztadat-halmazhoz nem feltétlenül léteznek elfogadó és generátor eljárások. Nem mindig létezik véges adekvát tesztadat-halmaz sem. Azonban minden esetben létezik

véges relatív adekvát tesztadat-halmaz, de ehhez nem mindig lehetséges elfogadó és generátor eljárásokat találni (4.2. Kapcsolódó kutatások és áttekintés).

A fejezetben az a célom, hogy releváns, a program futása során valóban előforduló tesztadatokat állítsak elő. Ezek ellenében lefuttatva az aktuális implementációt, nagy biztonsággal ellenőrizhető, hogy az megsérti-e valamely axiómát. Ennek érdekében kidolgoztam egy objektumorientált programok esetén alkalmazható naplózó eljárást, majd ezt a naplózó eljárást valósítottam meg a .NET programozási környezetben (4.5. A megvalósítás részletei). Végső soron a megoldás alkalmas a .NET által futtatott programok szekvenciapontonkénti, változó szintű naplózásra.

A naplózás alapú tesztadat generálást összekapcsoltam az axiomatikus teszteléssel, az axiómákat így képes vagyok releváns tesztadatok ellenében végrehajtani (4.7. A napló fájl és a generátor eljárás összekapcsolása).

3. tézis. Kidolgoztam egy napló formátumot, amely objektumorientált programozási nyelvek szekvenciapontonkénti naplózására használható. A .NET környezetben megvalósítottam a naplózó megoldást, amely a .NET CLR által futtatott programok esetén alkalmazható. A szekvenciapontonkénti naplózó eljárást összekapcsoltam a második tézisfejezetben bemutatott tesztadat generálással, amelynek eredményeként egy releváns tesztadatokat előállító eszközt kaptam.

Releváns publikációk

[13] M. Biczó, K. Pócza, I. Forgács, Z. Porkoláb. A New Concept of Effective Regression Test Generation in a C++ Specific Environment. Acta Cybernetica 18 (2008), pp. 481-501 2008.

[90] K. Pócza, M. Biczó, Z. Porkoláb. Cross-language Program Slicing in the.NET Framework. In : Conference proceedings of.NET Technologies 2005, Plzen (Czech Republic), pp. 141-150 2005.

[91] K. Pócza, M. Biczó, Z. Porkoláb. Towards Effective Runtime Trace Generation Techniques in the.NET Framework. In : Short communication papers proceedings of.NET Technologies 2006, Plzen (Czech Republic), pp. 9-16 2006.

[92] K. Pócza, M. Biczó, Z. Porkoláb. Towards detailed trace generation using the profiler in the.NET Framework. Annales Universitatis Scientiarum Budapestinensis de Rolando Eötvös Nominatae, Sectio Computatorica 30, pp. 21-40 2009.

Esettanulmány

Az ötödik fejezetben a felvázolt két fő irány szemléltetésére (hibák megelőzése helyességi specifikációval; releváns tesztadatokkal végrehajtott, az egységteszteknél magasabb szintű automatizált tesztek futtatása) bemutatok egy-egy konkrét példát. Ezzel az esettanulmánnyal azt kívánom megmutatni, hogy milyen módon alkalmazható a kétféle infrastruktúra nagyobb méretű alkalmazások esetében.

Kidolgozom a tesztalkalmazás kiválasztási szempontrendszerét, amely biztosítja, hogy a bemutatott példa kellően szemléletes legyen. Ez nem jelenti azt, hogy kizárólag a szempontrendszernek megfelelő alkalmazásokra alkalmazható eredményesen a kidolgozott eszköztár, azonban azokban az esetekben különösen könnyen felhasználható, amikor már a tervezésnél szem előtt tartjuk a tesztelhetőséget.

A kiválasztott tesztalkalmazás egy kiadásában ismertetek egy - a felhasználók munkáját megnehezítő - hibát. Megmutatom - amennyiben a fejlesztők alkalmazták volna a helyességi specifikációs eszköztárat -, hogyan lett volna elkerülhető a bejelentett hiba. Példát adok arra, hogy az axiomatikus tesztelés segítségével milyen lehetőség adódott volna egy egyszerű lokális strukturális átalakítás végrehajtására, és arra, hogy garantálni tudjuk, a hiba sem a jelenlegi, sem a későbbi implementációkban nem jelenik meg újra.

3. Kitekintés, továbbfejlesztési lehetőségek

A dolgozatban bemutatott eredmények nem jelentik ennek a kutatási témának a lezárását, a hatodik fejezetben megemlítem azokat a területeket, amelyeket további vizsgálatokra érdemesnek tartok. Ezek közül az egyik a specifikációs nyelvhez kapcsolódik. Érdemes lenne megvizsgálni, hogy az OCL modellező nyelv beépítése a helyességi specifikációs keretrendszerbe milyen előnyöket ígér, nyilván ez egy OCL → C# fordító használatát igényli majd.

A helyességi specifikációból levezetett alkalmazási terület, az axiomatikus tesztelés rámutatott arra, hogy ugyanennek az infrastruktúrának más felhasználása is

létezik. Érdemes lenne kísérleteket végezni folyamatok modellezésével, egy BPEL jellegű nyelv beépítésével. A kapott megoldás összehasonlítása a Windows Workflow Foundation szolgáltatásaival további témát kínál.

A harmadik terület, amely nélkül a kutatás nem tekinthető lezártnak: a releváns tesztadat-halmaz karbantartásának a problémája. A program kódja a különböző kiadások között (de akár két hibajavítás között is) számottevő változáson mehet keresztül. A valós tesztadatok generálása időigényes, emberi erőforrást igénylő feladat, és maga a tesztadatbázis a kód egy adott verziójára nézve érvényes. Fontosnak tartom, hogy bizonyos elemi program transzformációkhoz automatikusan frissíteni lehessen a tesztadatbázist annak teljes újraépítése nélkül. Ehhez a különféle dinamikus hatásvizsgálat-algoritmusok szolgáltatnak szilárd elméleti alapot.

Releváns publikációk összefoglaló táblázata

Az alábbi táblázat a felhasznált releváns publikációkat tartalmazza tézisenkénti bontásban (a sorokban a tézisek, az oszlopokban a publikációk a dolgozatban is alkalmazott sorszámai szerepelnek).

	[13]	[14]	[15]	[17]	[90]	[91]	[92]
I.		✓	✓	✓			
II.	✓			✓		✓	✓
III.	✓				✓	✓	✓